# [10 Examples of Optional in Java 8](https://javarevisited.blogspot.com/2017/04/10-examples-of-optional-in-java-8.html" \o "10 Examples of Optional in Java 8)

Null is bad, it can crash your program. Even it's creator called it a billion dollar mistake hence you should always try to avoid using nulls whenever you can. For example, you should not return a null String when you can return an empty String, similarly never return null collections when you can return an empty collection. I have shared many such tips in my earlier article, [10 tips to avoid NullPointerException](http://javarevisited.blogspot.sg/2013/05/ava-tips-and-best-practices-to-avoid-nullpointerexception-program-application.html) and my reader liked that a lot. But, I wrote that article a couple of years ago when Java 8 was not around and there was no Optional, a new way to avoid NullPointerException in Java, but, things has changed now. Java SE 8 style of coding is rapidly becoming the de-facto coding style in many companies and as a Java developer you should also learn and embrace the good things of Java 8 e.g. lambda expression, streams and of course the Optional.  
  
What is Optional? As the name suggests, the Optional is a wrapper class which makes a field optional which means it may or may not have values. By doing that, It improves the readability because the fact which was earlier hidden in the code is now obvious to the client.  
  
Though the idea of Optional is not new, in fact, Java SE 8 Optional is inspired from the ideas of Haskell and Scala. By using Optional, you can specify alternative values to return when something is null. For example, if you have an Employee Object and it has yet to assign a department, instead of returning null, you can return a default department. Earlier, there was no option to specify such default value in Java code but from Java 8 onwards, Optional can be used for that.

## How to create Optional Object in Java 8

There are many ways to create an object of the Optional class in Java, you can create an empty Optional by using the static method Optional.empty() as shown below:

Optional<Person> p = Optional.empty();

This Optional is empty, if you want to create an Optional with a non-null value then you can write following code:

Person p = new Person();

Optional<Person> op = Optional.of(p);

How this code is different from any code without Optional? Well, the biggest advantage of this code is that it will throw a [NullPointerException](http://www.java67.com/2012/09/nullpointerexception-in-java-nemesis-of.html" \t "_blank) immediately if Person p is null, rather than throwing a NullPointerException when you try to access any field of the person object.  
  
Third and probably the most useful way of creating an Optional instance is by using the ofNullable() method of java.util.Optional class which allows you to create an Optional object that may hold a null value as shown in the following example:

Optional<Person> op = Optional.ofNullable(p);

In case the Person object is null, the resulting Optional object would be empty, but it won't throw the NullPointerException.  
  
You can also create an instance of the Optional class by using the [static factory method](http://javarevisited.blogspot.sg/2017/02/5-difference-between-constructor-and-factory-method-in-java.html) Optional.of() in Java 8 as shown in the following example:

Address home = new Address(block, city, state, country, pincode);

Optional<Address> = Optional.of(home);

This code will return an Optional of Address object but value i.e. home must be not-null. In case the object is null then Optional.of() method will throw NullPointerException.

## How to use Optional to avoid NullPointerException in Java 8

Now, that you know how to create an Optional object let's see how to use it and find out whether it is better than the classical null check or not. Optional allows you to deal with the presence or absence of values instead of doing a [null check](http://javarevisited.blogspot.sg/2016/01/how-to-check-if-string-is-not-null-and-empty-in-java-example.html). Here is an example, suppose, we need to print the Person object if it is not null then this is how you used to write code before Java 8:

Person p = new Person("Robin", new Address(block, city, state, country);

Address a = p.getAddress();

if(a != null){

System.out.println(p);

}

Now, in Java 8 you can completely avoid this check by using the isPresent() method of the Optional class, which allows you to execute code if a value is printed and the code will not execute if no value is there, as shown in the following example:

Optional<Address> op = p.getAddress();

op.isPresent(System.out::println);

This is similar to how we use the [forEach()](http://javarevisited.blogspot.sg/2015/09/java-8-foreach-loop-example.html" \t "_blank) method before. Here the null check is enforced by the type system. If the Optional is empty i.e. person has no address then nothing would be printed.  
  
Btw, some Java programmer still uses the Optional like below:

if(!op.isPresent()){

System.out.println(p);

}

This is not recommended because it is similar to classical check and not the right way to use Optional in Java SE 8. You can further read [Java 8 in Action](http://www.amazon.com/Java-Action-Lambdas-functional-style-programming/dp/1617291994?tag=javamysqlanta-20) to learn more about the how to use Optional in Java SE 8.

## How to return a default value using Optional in Java 8

Now let see an example of how to return a default value if Optional is empty i.e. doesn't contain a value. You can use the Optional.orElse() method to return the default value as shown in the following example:

Person p = getPerson();

Address home = p.getAddress().orElse(Address.EMPTY);

Here the getAddress() method will return an Optional<Address> and that person doesn't have any address then orElse() method will return the empty address.  
  
You can also throw an exception if Optional doesn't contain any value by using the Optional.orElseThrow() method as shown below:

Address home = p.getAddress.orElseThrow(NoAddressException::new);

So, you can choose whatever your situation demands. Optional offers rich API to take different actions when a value is not present.

## How to use filter method with Optional in Java 8

Similar to the [Stream](http://javarevisited.blogspot.sg/2015/03/5-ways-to-convert-java-8-stream-to-list.html) class, Optional also provides a [filter()](http://javarevisited.blogspot.sg/2015/02/how-to-filter-collections-in-java-8.html) method to select or weed out unwanted values. For example, if you want to print all persons living in NewYork, you can write following code using the filter method of the Optional class:

Optional<Address> home = person.getAddress();

home.filter(address -> "NewYork".equals(address.getCity())

.ifPresent(() -> System.out.println("Live in NewYork"));

This code is safe because it will not throw any NullPointerException. This will print "Live in NewYork" if a person has address and city are equal to "NewYork". If the person doesn't have any address then nothing would be printed.  
  
Just compare this to the old style of writing safe code prior to Java 8 e.g. in JDK 6 or 7:

Address home = person.getAddress();

if(home != null && "NewYork".equals(home.getCity()){

System.out.println("NewYorkers");

}

The difference may not be significant in this case but as the chain of objects increases e.g. person.getAddress.getCity().getStreet().getBlock(), the first one will be more readable than the second one which will have to perform nested null checks to be safe. You can read [Java SE 8 for the Impatient](http://www.amazon.com/Java-SE8-Really-Impatient-Course/dp/0321927761?tag=javamysqlanta-20) to learn more about how to write functional code using Optional in Java.

## How to use map method with Optional in Java 8

The map() method of the Optional class is similar to the [map()](http://www.java67.com/2014/04/java-8-stream-examples-and-tutorial.html) function of Stream class, hence if you have used it before, you can use it in the same way with Optional as well. As you might know, map() is used to transform the object i.e. it can take a String and return an Integer. It applies a function to the value contained in the Optional object to transform it. For example, let's say you want to first check if person is not null and then want to extract the address, this is the way you would write code prior to Java 8

if(person != null){

Address home = person.getAddress();

}

You can rewrite this check-and-extract pattern in Java 8 using Optional's [map()](http://www.java67.com/2016/09/map-reduce-example-java8.html) method as shown in the following example:

Optional<Address> = person.map(person::getAddress);

Here we are using the [method reference](http://javarevisited.blogspot.sg/2017/03/what-is-method-references-in-java-8-example.html) to extract the Address from the Person object, but you can also use a [lambda expression](http://javarevisited.blogspot.sg/2014/02/10-example-of-lambda-expressions-in-java8.html) in place of method reference if wish. If you want to learn more about when you can use a lambda expression and when method reference is appropriate, you should a good book on Java 8. You can find some recommended Java 8 book [here](http://javarevisited.blogspot.sg/2016/10/best-books-to-learn-java-8.html).

## How to use flatMap method of Optional in Java 8

The flatMap() method of Optional is another useful method which behaves similarly to Stream.flatMap() method and can be used to replace unsafe cascading of code to a safe version. You might have seen this kind of code a lot while dealing with hierarchical objects in Java, particularly while extracting values from objects created out of XML files.

String unit = person.getAddress().getCity().getStreet().getUnit();

This is very unsafe because any of object in the chain can be null and if you check null for every object then the code will get cluttered and you will lose the readability. Thankfully, you can use the [flatMap()](http://javarevisited.blogspot.sg/2016/03/how-to-find-first-element-of-stream-in.html" \t "_blank) method of the Optional class to make it safe and still keep it readable as shown in the following example:

String unit= person.flatMap(Person::getAddress)

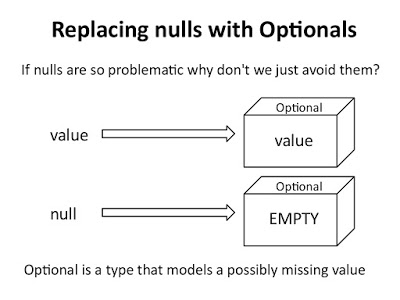
.flatMap(Address::getCity)

.flatmap(City::getStreet)

.map(Street::getUnit)

.orElse("UNKNOWN");

The first flatMap ensures that an Optional<Address> is returned instead of an Optional<Optional<Address>>, and the second flatMap does same to return an Optional<City> and so on.  
  
The important thing is the last call is a map() and not flatMap() because getUnit() returns a String rather than an Optional object. Btw, If you are confused between map and flatmap then I suggest you reading my earlier article [difference between map and flatmap in Java](http://javarevisited.blogspot.sg/2016/03/difference-between-map-and-flatmap-in-java8.html).

[](https://2.bp.blogspot.com/-qmVJGMs_KgI/WOuJM3jEM1I/AAAAAAAAISw/ojyP5T7Z3ew-g_Wf8XZaGIbZn8xZ1z3LACLcB/s1600/REplacing%2Bnulls%2Bwith%2Boptionals.jpg)

## Java 8 Optional Example

Here is the sample code for using Optional from Java 8. Optional can minimize the number of null checks you do in your code by explicitly saying that value can be null and set proper default values.

**package** test;

**import** **java.util.ArrayList**;

**import** **java.util.List**;

**import** **java.util.Optional**;

/\*\*

\* Simple example of how to use Optional from Java 8 to avoid [NullPointerException](http://javarevisited.blogspot.com/2012/06/common-cause-of-javalangnullpointerexce.html).

\* Optional is a new addition in Java API and also allows you to set default values for any object.

\*

\* @author Javin Paul

\*/

**public** **class** **OptionalDemoJava8**{

**public** **static** **void** **main**(String args[]) {

Address johnaddress = **new** Address("52/A, 22nd Street",

"Mumbai", "India", **400001**);

Person john = **new** Person("John",Optional.of(johnaddress), **874731232**);

Person mac = **new** Person("Mac", Optional.empty(), **333299911**);

Person gautam = **new** Person("Gautam", Optional.empty(), **533299911**);

List<Person> people = **new** ArrayList<>();

people.add(john);

people.add(mac);

people.add(gautam);

people.stream().forEach((p) -> {

System.out.printf("%s from %s %n",

p.name(),

p.address().orElse(Address.EMPTY\_ADDRESS));

});

}

}

**class** **Person**{

**private** String name;

**private** Optional<Address> address;

**private** **int** phone;

**public** **Person**(String name, Optional<Address> address, **int** phone) {

**if**(name == **null**){

**throw** **new** **IllegalArgumentException**("Null value for name is not permitted");

}

**this**.name = name;

**this**.address = address;

**this**.phone = phone;

}

**public** String **name**(){

**return** name;

}

**public** Optional<Address> **address**(){

**return** address;

}

**public** **int** **phone**(){

**return** phone;

}

**@Override**

**public** String **toString**() {

**return** "Person{" + "name=" + name + ", address=" + address

+ ", phone=" + phone + '}';

}

}

**class** **Address**{

**public** **static** **final** Address EMPTY\_ADDRESS = **new** Address("","","",**0**);

**private** **final** String line1;

**private** **final** String city;

**private** **final** String country;

**private** **final** **int** zipcode;

**public** **Address**(String line1, String city, String country, **int** zipcode) {

**this**.line1 = line1;

**this**.city = city;

**this**.country = country;

**this**.zipcode = zipcode;

}

**public** String **line1**(){

**return** line1;

}

**public** String **city**(){

**return** city;

}

**public** String **country**(){

**return** country;

}

**public** **int** **zipcode**(){

**return** zipcode;

}

**@Override**

**public** String **toString**() {

**return** "Address{" + "line1=" + line1 + ", city=" + city + ", country=" + country + ", zipcode=" + zipcode + '}';

}

}

**Output:**

John from Address{line1=**52**/A, **22**nd Street, city=Mumbai, country=India, zipcode=**400001**}

Mac from Address{line1=, city=, country=, zipcode=**0**}

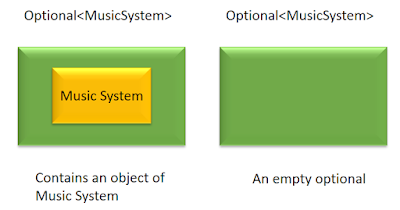
Gautam from Address{line1=, city=, country=, zipcode=**0**}

If you're going to use null, consider the @Nullable annotation. Many Java IDEs like IntelliJ IDEA has built-in support for the @Nullable annotation. They also show nice inspections as shown below to prevent you from using Optional in the wrong way.

[](https://2.bp.blogspot.com/-dB9O5u0rtMk/WOuJq1AehdI/AAAAAAAAIS0/xWlSzEM8zEs-TsdG2TLN3Vbu_XYt_vyXQCLcB/s1600/Java%2B8%2BOptional%2Bexample.png)

## Important points about Optional class in Java 8

Here are some of the key points about the java.util.Optional class which is worth remembering for future use:  
  
1) The Optional class is a container object which may or may not contains a non-null value.  That's why it is named Optional.  
  
2) If a non-value is available then Optional.isPresent() method will return true and get() method of Optional class will return that value.  
  
3) The Optional class also provides methods to deal with the absence of value e.g. you can call Optional.orElse() to return a default value if a value is not present.  
  
4) The java.util.Optional class is a value-based class and use of identity sensitive operations e.g. using the [== operator](http://www.java67.com/2012/11/difference-between-operator-and-equals-method-in.html), calling [identityHashCode()](http://www.java67.com/2016/09/difference-between-identityhashmap-weakhashmap-enummap-in-java.html" \l "uds-search-results" \t "_blank) and [synchronization](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html) should be avoided on an Optional object.  
  
5) You can also use the orElseThrow() method to throw an exception if a value is not present.  
  
6) There are multiple ways to create Optional in Java 8. You can create Optional using the static factory method Optional.of(non-null-value) which takes a non-null value and wrap it with Optional. It will throw NPE if the value is null. Similarly, the Optional.isEmpty() method return an empty instance of Optional class in Java.

[](https://3.bp.blogspot.com/-J8ISpH21Mck/WOuK_lYudyI/AAAAAAAAITA/t0f1Fy-BuUQK3owm9eLTFLBjT36j7VNhACLcB/s1600/Optional%2Bin%2BJava.png)

7) The biggest benefit of using Optional is that it improves the readability and convey information which fields are optional, for example, if you have a Computer object you can put CD drive as optional because now a day modern laptops like HP Envy doesn't have a CD or Optical drive.  
  
Earlier it wasn't possible to convey client which fields are optional and which are always available, but now, if a getter method return Optional than client knows that value may or may not be present.  
  
8) Similar to java.util.stream.Stream class, Optional also provides [filter()](http://www.java67.com/2016/08/java-8-stream-filter-method-example.html), [map()](http://www.java67.com/2015/01/java-8-map-function-examples.html), and [flatMap()](http://www.java67.com/2016/03/how-to-use-flatmap-in-java-8-stream.html) method to write safe code in Java 8 functional style. The method behaves similarly as they do in Stream class, so if you have used them before, you can use them in the same way with the Optional class as well.  
  
9) You can use the map() method to transform the value contained in the Optional object and flatMap() for both transformations and flattening which would be required when you are doing the transformation in a chain as shown in our Optional + flatMap example above.  
  
10) You can also use the filter() method to weed out any unwanted value from the Optional object and only action if Optional contains something which interests you.

Read more: <https://javarevisited.blogspot.com/2017/04/10-examples-of-optional-in-java-8.html#ixzz6Ftfw33bs>